

Hardware-assisted Virtualization on non-Intel Processors

Alexander Deibel, Florian Kargl, Stefan Weiglhofer, Hannes Weissteiner

May 10, 2021

Overview of virtualization techniques on various architectures

- AMD
- ARM
- RISC-V

AMD

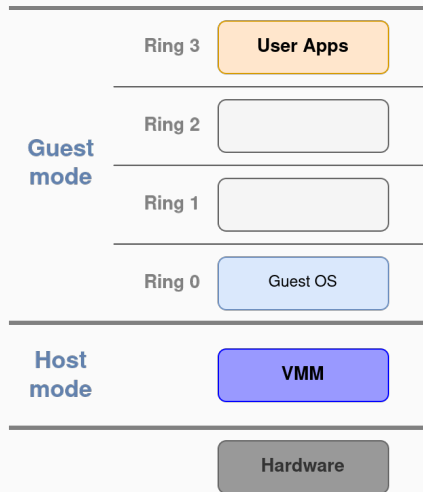
AMD Virtualization (AMD-V)

- Released in 2006 for Athlon 64 series
- Basic functionality supported by all Zen-based AMD processors
- Some extensions (e.g. AMD-SEV) only for higher-end/server models
- Similar concepts to Intel VT-x



AMD-V Architecture

- Adds new privilege mode exclusively for hypervisor use
- VMM in "ring -1"
- Two operation modes:
 - Host mode
(similar to VMX Root Operation)
 - Guest mode
(similar to VMX Non-Root Operation)



Setting `EFER.SVME` to 1 enables the following instructions (similar to `VMXON` instruction on Intel):

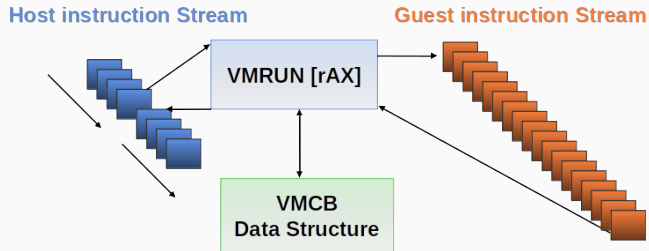
- `VMRUN` - to enter guest mode
- `VMLOAD/VMSAVE` - saves/restores additional guest state information
- `CLGI/STGI` - sets/clears the global interrupt flag (GIF)
- `INVLPGA` - allows to selectively invalidate TLB mappings using a given ID (ASID)
- `VMMCALL` - a way for a guest to explicitly call the VMM
- `SKINIT` - used to verify and load trusted software (e.g. a VMM)

Virtual Machine Control Block (VMCB)

- Data structure (similar to VMCS on Intel)
- Contains:
 - Control bits to define the guest exit behaviour
 - Control bits that specify the execution environment (e.g. nested paging)
 - A guest processor state (e.g. control registers, ...)
- VMRUN/VMEXIT
only saves/restores minimal amount of state information
- VMLOAD/VMSAVE
used to load/store additional state information (e.g. hidden processor states, ...)

Entering VMM mode

- Enter guest mode with VMRUN instruction (comparable to VMLAUNCH/VMRESUME)
- Guest runs until a #VMEXIT occurs



Exiting VMM mode (#VMEXIT)

- Guest runs until intercepts happens:
 - Exception or Interrupt
 - Instruction (e.g. VMSCALL)
- Information about the exit reason is put into the VMCB
- Advanced Virtual Interrupt Controller (AVIC)
 - Reduction of interrupt overhead for virtualization

AMD-V Further Features

- **Second Level Address Translation**

Nested Page tables (NPT)/Rapid Virtualization Indexing (RVI) (EPT on Intel):

- Enable by setting NP_ENABLE bit in the VMCB to 1
- Set NPT base pointer via N_CR3 in the VMCB

- **TLB Control**

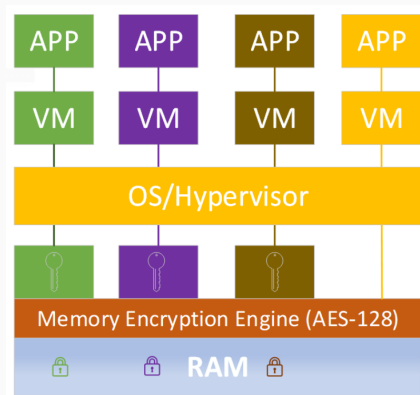
TLB entries are tagged with Address Space Identifier (ASID) to differentiate between guest physical address spaces

- **I/O-Virtualization**

AMD-Vi: enables virtualization of I/O-devices through the use of DMA and interrupt remapping

AMD-V Security Extensions

- **Secure Encrypted Virtualization (SEV)**
Guards against guest memory inspection by assigning a unique AES encryption key to automatically encrypt their in-use data
- **SEV Encrypted State (SEV-ES)**
Guest register state is encrypted on each hypervisor transition
- **SEV Secure Nested Paging (SEV-SNP)**
Adds memory integrity protection



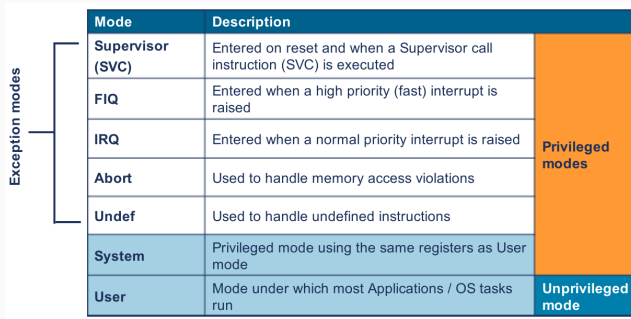
ARM

Not too long ago in a galaxy not all that far away...

ARMv7-A without extensions

(~2005-2011)

- Lots of different processor modes
 - Different registers available
 - Different stack space
 - Determines privilege level
- Some operations only available in privileged modes
- No virtualization support



Mode	Description	
Supervisor (SVC)	Entered on reset and when a Supervisor call instruction (SVC) is executed	Privileged modes
FIQ	Entered when a high priority (fast) interrupt is raised	
IRQ	Entered when a normal priority interrupt is raised	
Abort	Used to handle memory access violations	
Undef	Used to handle undefined instructions	
System	Privileged mode using the same registers as User mode	Unprivileged mode
User	Mode under which most Applications / OS tasks run	

Figure 1: ARMv7-A processor modes

Popek and Goldberg virtualization requirements

- **Equivalence**

Guest software behaves identical to native execution

- **Resource control**

Guest software is not allowed to access physical state and resources

- **Efficiency**

All non-sensitive instructions are executed natively without VMM intervention

- **Sensitive instructions**

Instructions that change/read system state, access physical resources, ...

- **Privileged instructions**

Privileged instructions are always trapped into a privileged mode when executed in an unprivileged mode.

Popek and Goldberg theorem

If the set of sensitive instructions is a subset of privileged instructions, a system can be efficiently virtualized.

⇒ Trap-and-emulate virtualization

ARM virtualization challenges

Problems with ARMv7-A:

- Not all sensitive instructions are privileged and cause a trap, e.g.
 - Interaction with coprocessors (modify system state)
 - Wait for interrupt
 - Return from event handlers (change processor mode)

⇒ ARMv7-A can *not* be virtualized via trap-and-emulate!

But: Not all is lost, virtualization still possible via dynamic binary translation

- VMM interprets guest code at runtime and emulates sensitive instructions
- Unfortunately quite slow...

Full system virtualization not used very often for ARMv7-A

⇒ Paravirtualization to the rescue

Solving the ARM virtualization problem

Virtualization Extensions introduced for the ARMv7-A architecture in 2011 to solve these problems.

- New *Hyp* mode with higher privilege level 2
- Allow sensitive instructions to be trapped into the hypervisor mode
- Two stage address translation for VMs (similar to Intel EPT)
- IRQs and exceptions can be routed to the hypervisor + virtual IRQ injection
- HVC hypervisor call instruction (for paravirtualization)
- Virtualization support for standard peripherals (interrupt controller, timer)
- No dedicated VM control block in memory. State needs to be saved/restored by the hypervisor.

ARMv7-A Virtualization Extensions

The diagram shows a table of ARMv7-A modes. A bracket on the left groups the first six modes (Supervisor (SVC), FIQ, IRQ, Abort, Undef, System) as 'Exception modes'. A large orange block on the right groups the first six modes as 'Privileged modes' and the last two modes (System, User) as 'Unprivileged mode'.

Mode	Description	Category
Supervisor (SVC)	Entered on reset and when a Supervisor call instruction (SVC) is executed	Privileged modes
FIQ	Entered when a high priority (fast) interrupt is raised	
IRQ	Entered when a normal priority interrupt is raised	
Abort	Used to handle memory access violations	
Undef	Used to handle undefined instructions	
System	Privileged mode using the same registers as User mode	
User	Mode under which most Applications / OS tasks run	Unprivileged mode

Figure 2: ARMv7-A modes

The diagram shows a table of ARMv7-A modes with virtualization extensions. A bracket on the left groups the first seven modes (Supervisor (SVC), FIQ, IRQ, Abort, Undef, Hyp, Monitor) as 'Exception modes'. A bracket on the left groups the last three modes (System, User, and an unlabeled mode) as 'Same register bank'. A large yellow block on the right groups the first seven modes as 'Privileged modes' and the last two modes (System, User) as 'Unprivileged mode'.

Mode	Description	Category
Supervisor (SVC)	Entered on reset and when a Supervisor call instruction (SVC) is executed	Privileged modes
FIQ	Entered when a high priority (fast) interrupt is raised	
IRQ	Entered when a normal priority interrupt is raised	
Abort	Used to handle memory access violations	
Undef	Used to handle undefined instructions	
Hyp	Used for hardware virtualization support	
Monitor	Used for TrustZone secure monitor program	
System	Privileged mode using the same registers as User mode	Unprivileged mode
User	Mode under which most Applications / OS tasks run	

Figure 3: ARMv7-A modes with virtualization extension

ARMv7-A Virtualization Extensions - Privilege Levels

'Un-/privileged' renamed to privilege level (PL)

- **PL0**: User applications
- **PL1**: (Guest) kernel
- **PL2**: Hypervisor

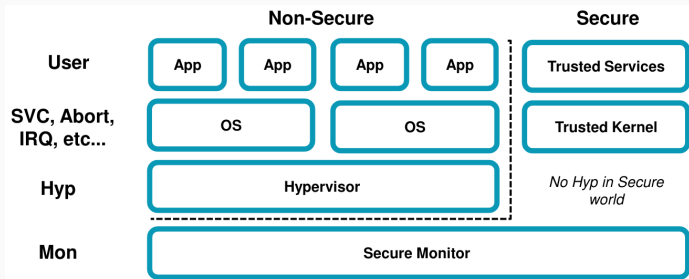


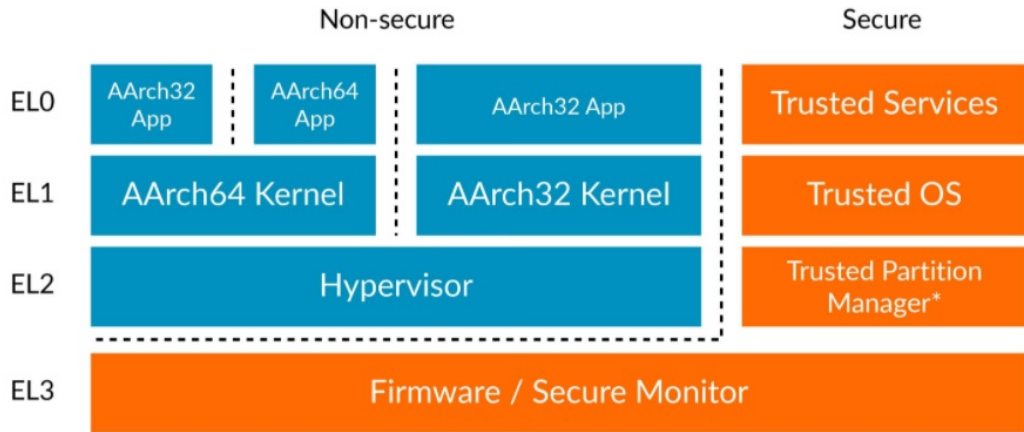
Figure 4: ARMv7-A virtualization extension privilege levels

- Exception Levels
- Execution states
- Memory Management



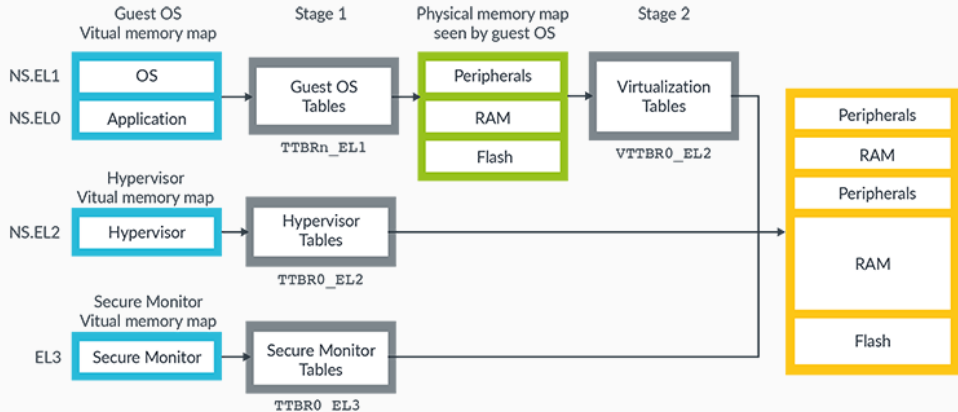
Figure 5: Arm

Armv8-A overview



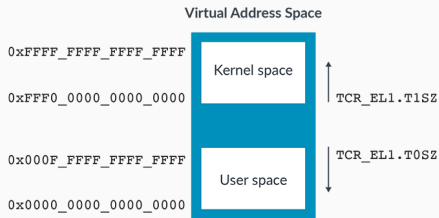
* Secure EL2 from Armv8.4-A

Armv8-A Virtual Memory



EL0/EL1 virtual address space is split in two parts

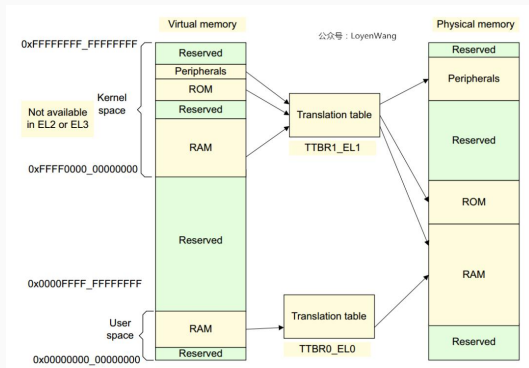
- Low virtual addresses (userspace)
 - Size configurable via `TCR_EL1.T0SZ`
 - Page table base `TTBR0_EL1`
- High virtual addresses (kernel)
 - Size configurable via `TCR_EL1.T1SZ`
 - Page table base `TTBR1_EL1`



ARMv8-A - Virtual Memory

EL0/EL1 virtual address space is split in two parts

- Low virtual addresses (userspace)
 - Size configurable via `TCR_EL1.T0SZ`
 - Page table base `TTBR0_EL1`
- High virtual addresses (kernel)
 - Size configurable via `TCR_EL1.T1SZ`
 - Page table base `TTBR1_EL1`

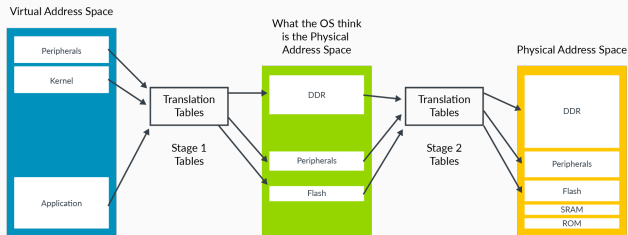


ARMv8-A Virtualization Extensions - Virtual Memory

EL0/1 (Guest OS)

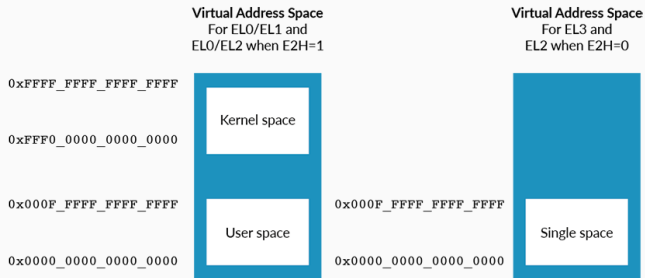
- Two stage address translation

- Guest virtual address
 - Intermediate physical address
 - Host physical address
 - Set stage 2 page table base via VTTBR_EL2
- Enable via HCR_EL2.VM
- TLB entries tagged with VMID from VTTBR_EL2



EL2 (Hypervisor)

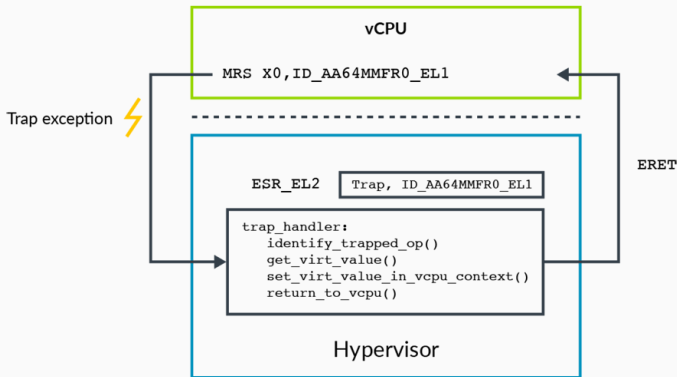
- Separate page tables for EL2
 - Base address in TTBR0_EL2
- Only low virtual addresses available (no TTBR1_EL2)



ARMv8-A Virtualization Extensions - Trapping operations

Instructions to be trapped
configurable in HCR_EL2

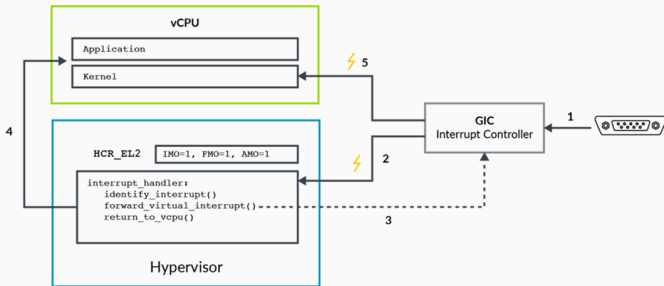
1. Trap triggers exception into EL2
2. Inspect ESR_EL2 for exception reason
3. Modify guest state
4. Return to guest via ERET



ARMv8-A Virtualization Extensions - Exception routing

Exceptions/IRQs/FIQs can be intercepted by hypervisor

- Enabled via `HCR_EL2.IMO`
 - All IRQs routed to EL2 instead of EL0/EL1
 - Hypervisor can send virtual interrupts to guest by setting `HCR_EL2.VI`



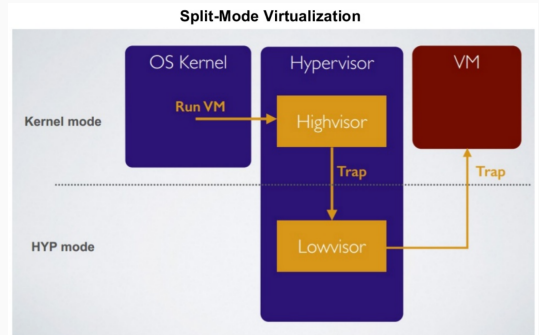
Setting up virtualization

- Allocate space for guest state
- Setup Hypervisor Configuration register
HCR_EL2
 - Trapped instructions
 - Exception routing
- Set up stage 2 translation tables in VTTBR_EL2
 - Page table base
 - VMID
- Setup EL1/EL0 registers for guest
- Execute ERET to return to guest VM
- Wait for hypervisor trap

Problems with ARM Virtualization Extension

The ARM virtualization extensions are great for type 1 (bare metal) hypervisors, but not so much for type 2 (hosted) hypervisors.

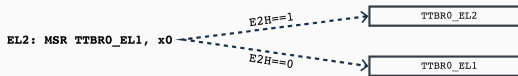
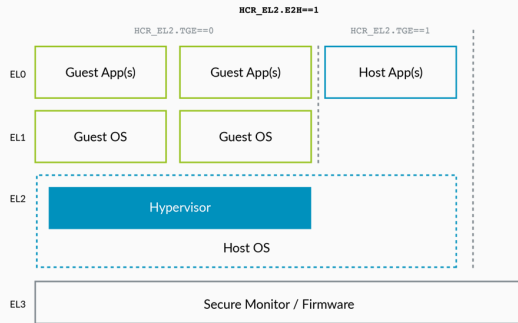
- Cannot run a kernel designed for EL1 in EL2. EL2 is not a superset of EL0/EL1 features.
 - Different system registers in EL1/EL2
 - Different virtual address space
 - Only low virtual addresses usable in EL2 (conflicts with userspace)
- Host kernel in EL1 + small hypervisor shim in EL2



ARMv8.1 - Virtualization Host Extensions

ARM Virtualization Host Extensions allow unmodified EL1 kernels to run in EL2 (good for type 2 hosted hypervisors)

- Lower+upper virtual address regions in EL2, similar to EL1
 - Enable via `HCR_EL2.E2H`
 - Set `HCR_EL2.TGE` when running host applications to route all exceptions to EL2
- Automatically redirect system register access to EL2 registers
 - EL1 registers still available as `<reg>_EL12`



ARMv8.4 - Nested Virtualization

- Guest Hypervisors can't run in EL2
- ARMv8.3: Trap accesses to `_EL2` registers
- Process the requested access in EL2

ARMv8.4 - Nested Virtualization

- Guest Hypervisors can't run in EL2
- ARMv8.3: Trap accesses to `_EL2` registers
- Process the requested access in EL2

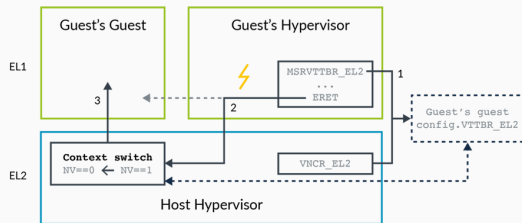
But that's slow

ARMv8.4 - Nested Virtualization

- Guest Hypervisors can't run in EL2
- ARMv8.3: Trap accesses to `_EL2` registers
- Process the requested access in EL2

But that's slow

- Solution:
 - Capture the state of Guest-`_EL2` registers
 - State location in `VNCR_EL2`
 - Handle on `ERET`
- Controlled by `HCR_EL2.NV*`-bits



RISC-V

Hypervisor Extension, V0.6.1

- Draft Version 0.6.1, not yet accepted as standard
- Hosting of guest OS atop type-1 (bare-meta) or type-2 (hosted) hypervisor
- Focused on CPU Virtualization
 - Full duplicate of the CPU state (new and shadow CSRs)
 - Two-Stage Address Translation (enabled when $V = 1$)
 - No dedicated I/O virtualization specified



Privilege Modes

- S-mode changed to HS-mode
- In VU- and VS-mode $V = 1$
- HS has higher interrupt priority than VS

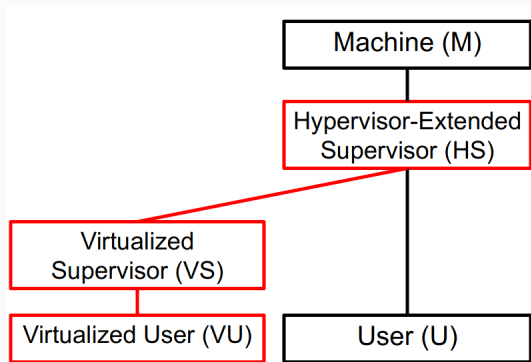


Figure 6: RISC-V Privilege Levels

New Registers

- Changed Machine Level Regs (added MPV GVA fields to `mstatush`)
- Hypervisor status `hstatus`
- `hedeleg` & `hideleg` delegate traps to VS-Mode guest
- Interrupt & Timing registers
- Trap registers `htval`, `htinst`
- Guest stage translation `hgap`
- Accesses to following registers substitute to respective shadow registers (e.g. access to `sstatus` is directed to `vsstatus`)
 - `sstatus`, `sip`, `sie`, `stvec`, `sscratch`, `sepc`, `scause`, `stval`, `satp`

- Virtual-Machine Load and Store
 - only in M-mode or HS-mode
 - access guest virtual address space
 - inspect guest memory without mapping it
- Privileged Fence
 - Applies to the new memory spaces
 - Structures controlled by `vsatp` or `hvatp`

Two-Stage Address Translation

- Virtual address converted to guest physical address (VS-stage)
- Guest physical address to supervisor physical address (G-stage)
- Root page table expanded by factor four to 16KiB
- Same format as single-stage address translation

Reference Implementation

- Rocket chip core
- Ported 'Bao' Hypervisor from ARM to RISC-V
- Optimized PLIC and CLINT
- Still no IOMMU, would improve DMA accesses (no traps to HS-mode)

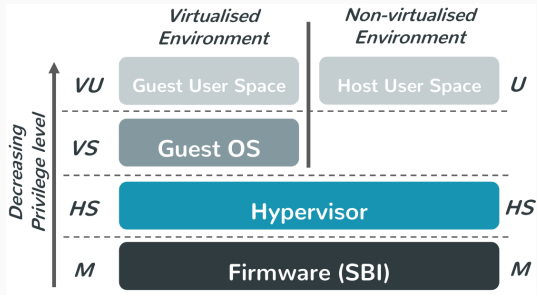


Figure 7: RISC-V Hypervisor and Guest OS Privileges

- [1] AMD64 Architecture Programmer's Manual. *Volume 2: System Programming*. <https://www.amd.com/system/files/TechDocs/24593.pdf>. Online; accessed 25 April 2021.
- [2] Keith Adams et. al. *A Comparison of Software and Hardware Techniques for x86 Virtualization*. https://www.vmware.com/pdf/asplos235_adams.pdf. Online; accessed 25 April 2021.
- [3] Fengwei Zhang et. al. *A comparison study of intel SGX and AMD memory encryption technology*. <https://dl.acm.org/doi/pdf/10.1145/3214292.3214301>. Online; accessed 25 April 2021.

- [4] Niels Penneman et. al. *Formal virtualization requirements for the ARM architecture*. <https://users.elis.ugent.be/~brdsutte/research/publications/2013JSApenneman.pdf>. Online; accessed 25 April 2021.
- [5] ARM Architecture Reference Manual ARMv7-A. *ARMv7-A processor modes*. <https://developer.arm.com/documentation/ddi0406/b/System-Level-Architecture/The-System-Level-Programmers--Model/ARM-processor-modes-and-core-registers/ARM-processor-modes?lang=en>. Online; accessed 25 April 2021.

- [6] ARM Architecture Reference Manual ARMv7-A. *ARMv7-A privilege levels*. <https://developer.arm.com/documentation/ddi0406/c/Application-Level-Architecture/Application-Level-Memory-Model/Access-rights/Processor-privilege-levels--execution-privilege--and-access-privilege?lang=en>. Online; accessed 25 April 2021.
- [7] ARM Developers Guide. *ARMv8-A exception model*. <https://developer.arm.com/documentation/102412/0100/Privilege-and-Exception-levels>. Online; accessed 25 April 2021.

- [8] ARM Developers Guide. *AArch64 Exception model*. <https://developer.arm.com/documentation/102412/0100/Execution-and-Security-states>. Online; accessed 25 April 2021.
- [9] ARM Developers Guide. *AArch64 memory management*. <https://developer.arm.com/documentation/101811/0100/Address-spaces-in-AArch64>. Online; accessed 25 April 2021.
- [10] ARM Developers Guide. *AArch64 Virtualization*. <https://developer.arm.com/documentation/102142/latest>. Online; accessed 25 April 2021.

- [11] Andrew Waterman et. al. *The RISC-V Instruction Set Manual Volume II: Privileged Architecture*. <https://github.com/riscv/riscv-isa-manual/releases/tag/draft-20210402-1271737>. Online; accessed 25 April 2021.
- [12] Bruno Sà et. al. *A First Look at RISC-V Virtualization from an Embedded Systems Perspective*. <https://arxiv.org/pdf/2103.14951.pdf>. Online; accessed 25 April 2021.
- [13] Andrew Waterman et. al. *RISC-V Hypervisor Extension*. <https://riscv.org/wp-content/uploads/2017/12/Tue0942-riscv-hypervisor-waterman.pdf>. Online; accessed 25 April 2021.